TEXT-BASED ADVENTURE GAME

OVERVIEW

In this project, your team will develop a text-based adventure game. The setting for your game will be decided by your team, but examples include:

- Fantasy (knights, dragons, elves, castles, etc.)
- Space (astronauts, aliens, spaceships, etc.)
- Contemporary (current society and locations)
- Nostalgic (deep-dark Africa explorers, 1920's hard-boiled detective, haunted house)

REQUIREMENTS

Your text-based adventure game is required to meet the following minimum requirements:

- Multiple environments. The game must present the character with different environments to traverse (e.g. A cave, a castle, a forest, or different types of rooms in a haunted house).
- Multiple characters. The game will have supporting characters (non-playable characters) that the main character interacts with (e.g. A king that gives a knight a quest, a woman that hires detective, a bridge guardian that asks questions of a traveler)
- Multiple actions. The main character can interact with an environment or items in different ways (e.g. Movement, fighting, using items)
- Multiple usable items. A usable item allows the main character to interact with another object in some way (e.g. a key to open a door or a chest). Not all usable items need to be useful in the game.
- A basic plot. The basic plot of the game will involve an end goal requiring multiple steps (min 10 steps) to achieve (e.g. Rescue a princess from a dragon, discover who stole a priceless statue, and defeat an alien invasion on a contested planet). There must also be ways that player can lose the game (e.g. Eaten by a dragon, shot by the police, didn't prevent an alien invasion in time).
- The ability to save and restore progress. Your game must allow a user to quit the game and then resume from where they left off (or as close to where they were in the game as is reasonable).
- Provide means for getting help when playing the game (e.g. "help" prints out the list of all possible actions).
- Error-checking to prevent program crashing (e.g. Validating all input from the user, ignoring/warning about nonsensical actions)

PROJECT PHASES

The project will have a number of phases:

1. *Design* where the game will be designed and the project planned.
2. *Implementation* where the game is implemented.
3. *Maintenance* where your team will test a game developed by another team, and your team will address feedback from the testers.

PROCESS CONSTRAINTS

To help keep the project manageable across the different teams, you will have the following process constraints:

USER INTERFACE

The game will be implemented as a text game (i.e. no GUI) with all interaction on the command-line. If you would like to use ASCII art, that is fine, **however** the art must be found in one or more separate files that are read in to keep the code readable (this is also good SE practice).

SOFTWARE TOOLS

1. The project is to be developed in **C++.**
2. The team will use a **public** Mercurial repository on **Bitbucket** for version control and issue tracking.

PLATFORM

1. The project will run in the Linux environment of the University of Lethbridge computer science labs.

REPOSITORY ORGANIZATION

Your repository must be organized in a logical fashion (i.e. do not have all files at the top level). Your repository is required to have at least the following top-level directories and files (so the grader can easily find the files and build your project), but you can add other directories according to your project needs:

- `Makefile` – a makefile that has the following targets:
  - `compile` – compiles the project
  - `test` – compiles and runs the unit test cases
  - `memory` – runs `valgrind` on the project
  - `coverage` – runs `gcov` on the project
  - `docs` – generates the code documentation using `doxygen`
- `project –` Code::Block project files `(.cbp, .depend, .layout)`, if used
- `src` - the implementation files (`.cpp`), including the `main.cpp` to run the program
- `include` – the header files (`.h`)
- `test` – the cppUnit test files, including the `main.cpp` to run the tests
- `docs` – contains the project documentation, with the following sub-directories:
  - `design` - design document and UML diagrams
  - `code` - source code documentation (doxygen)
  - `user` - user manual
  - `testing` – testing and maintenance report
  - `team` – the team reports
    - `design` – documents from the design phase
    - `implementation` – documents from the implementation phase
    - `maintenance` – documents from the testing & maintenance phase